United Electronic Industries

The High-Performance Alternative

DNx-SL-504

—

User Manual

4-Channel RS-232 or RS-422/423/485 (serial port) layer for the PowerDNA Cube and PowerDNR RACKtangle with support for Synchronous Serial Data Communcation protocols (SDLC, HDLC, async, bisync protocols)

Release 4.5

November 2012

PN Man-DNx-SL-504-1112

© Copyright 1998-2012 United Electronic Industries, Inc. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form by any means, electronic, mechanical, by photocopying, recording, or otherwise without prior written permission.

Information furnished in this manual is believed to be accurate and reliable. However, no responsibility is assumed for its use, or for any infringement of patents or other rights of third parties that may result from its use.

All product names listed are trademarks or trade names of their respective companies.

See the UEI website for complete terms and conditions of sale:

http://www.ueidaq.com/cms/terms-and-conditions/

Contacting United Electronic Industries

Mailing Address:

27 Renmar Avenue
Walpole, MA 02081
U.S.A.

For a list of our distributors and partners in the US and around the world, please see
http://www.ueidaq.com/partners/

Support:

Telephone: (508) 921-4600
Fax: (508) 668-2350

Also see the FAQs and online “Live Help” feature on our web site.

Internet Support:

Support: support@ueidaq.com
Web-Site: www.ueidaq.com
FTP Site: ftp://ftp.ueidaq.com

Product Disclaimer:

WARNING!

DO NOT USE PRODUCTS SOLD BY UNITED ELECTRONIC INDUSTRIES, INC. AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS.

Products sold by United Electronic Industries, Inc. are not authorized for use as critical components in life support devices or systems. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness. Any attempt to purchase any United Electronic Industries, Inc. product for that purpose is null and void and United Electronic Industries Inc. accepts no liability whatsoever in contract, tort, or otherwise whether or not resulting from our or our employees' negligence or failure to detect an improper purchase.

Specifications in this document are subject to change without notice. Check with UEI for current status.

Table of Contents

Chapter 1 Introduction 1
1.1 Organization of Manual 1
1.2 The SL-504 Interface Board 3
1.3 Features 4
1.4 Indicators 4
1.5 Specification 5
1.6 What is Serial Communication? 6
1.7 OSI Layer 1 and Layer 2 Coordination 7
1.8 What is Synchronous Serial Communica-tion? 8
1.9 Architecture 12
1.10 Module Capabilities 13
1.11 Wiring & Connectors 14
1.12 Jumper Settings for Board Position 15
Chapter 2 Programming with the High Level API 16
2.1 Creating a Session 16
2.2 Configuring the Resource String 16
2.2.1 Asyncronous serial port 16
2.2.2 HDLC Port 18
2.3 Configuring the Timing 20
2.4 Read Data 20
2.4.1 Asynchronous Mode 20
2.4.2 HDLC Mode 20
2.5 Write Data 21
2.5.1 Asynchronous Mode 21
2.5.2 HDLC Mode 21
2.6 Cleaning-up the Session 21
Chapter 3 Programming with the Low-level API 22
3.1 HDLC/SDLC Mode 23
3.2 Asynchronous Mode 24
3.2.1 Setting the configuration 24
3.3 NRZI Mark Encoding 27
3.3.1 Enabling Operations 30
3.3.2 Sending and Receiving HDLC Frames 31
3.3.3 Sending and Receiving Asynchronous Data 31
3.3.4 Obtaining Link Status 32
3.3.5 Aborting Transmission 34

© Copyright 2012
United Electronic Industries, Inc.

List of Figures

1-1 The DNA-SL-504 Serial Layer 4
1-2 RS-485 Topologies 6
1-3 UART Data Frames for RS-232 and RS-485 7
1-4 Logic Block Diagram: DNA/DNR-SL-504 Overview 12
1-5 DNx-SL-504 Connection Diagram 14
1-6 Jumper Block for DNA-SL-504 Board Position 15
1-7 Diagram of DNA-SL-504 Board Position Jumper Settings 15
A-1 Pinout and photo of DNA-STP-62 screw terminal panel 35

© Copyright 2012
United Electronic Industries, Inc.

Chapter 1 Introduction

This document outlines the feature-set of the DNR- and DNA-SL-504 layer and its use for synchronous serial-line communications applications.

1.1 Organization of Manual

This SL-504 User Manual is organized as follows:

- **Introduction**
  This section provides an overview of the SL-504 synchronous serial line communication interface features, device architecture, and connectivity.
- **Programming with the High-Level API**
  This chapter provides an overview of the how to create a session, configure the session, and format relevant data with the Framework API.
- **Programming with the Low-Level API**
  Describes low-level API commands for configuring and using the SL-504 series layer for serial operating modes.
- **Appendix A - Accessories**
  This appendix provides a list of accessories available for use with the DNx-SL-504 serial-line communication interface board.
- **Index**
  This is an alphabetical listing of the topics covered in this manual.

© Copyright 2012
United Electronic Industries, Inc.

Manual Conventions

To help you get the most out of this manual and our products, please note that we use the following conventions:

Tips are designed to highlight quick ways to get the job done or to reveal good ideas you might not discover on your own.

NOTE: Notes alert you to important information.

CAUTION! *Caution advises you of precautions to take to avoid injury, data loss, and damage to your boards or a system crash.*

Text formatted in **bold** typeface generally represents text that should be entered verbatim. For instance, it can represent a command, as in the following example: “You can instruct users how to run setup using a command such as **setup.exe**.”

Text formatted in `fixed` typeface generally represents source code or other text that should be entered verbadim into the source code, initialization, or other file.

Examples of Manual Conventions

Before plugging any I/O connector into the Cube or RACKtangle, be sure to remove power from all field wiring. Failure to do so may cause severe damage to the equipment.

Usage of Terms

Throughout this manual, the term “Cube” refers to either a PowerDNA Cube product or to a PowerDNR RACKtangle™ rack mounted system, whichever is applicable. The term DNR is a specific reference to the RACKtangle, DNA to the PowerDNA I/O Cube, and DNx to refer to both.

© Copyright 2012
United Electronic Industries, Inc.

1.2 The SL-504 Interface Board

The DNA- and DNR-SL-504 layer is a 4-port serial communications interface Cube/RACKtangle I/O chassis respectively. Each port is independently configurable as RS-232, RS-485, RS-422/423 as well as set for synchronous or asynchronous communications. Each port is fully isolated from the other three ports as well as from the Cube or RACKtangle chassis. The board is an ideal interface to a wide variety of serial based data acquisition and control interfaces as well as general purpose serial I/O.

The DNx-SL-504 is based on the Zilog Z16C32 serial controller chip and supports most popular synchronous serial protocols including HDLC and SDLC. The HDLC/SDLC interface provides full access to the serial frames. User code can then determine how to handle retry or protocol corrections. The RS-485/422 implementation provides transmit and receive data, synch and clock interfaces. The RS-232 configuration also supports CTS signals. The maximum transfer rate in RS-485/422 and RS-232 modes are 4Mbaud and 600kbaud in synchronous mode respectively.

The DNx-SL-504 support standard asynchronous serial transfer rates up to 2 Mbaud in RS-485 mode or up to 256 kbaud in RS-232 mode. It also supports communications at 12, 12.5 and 50 kbaud with better than 0.1% data rate accuracy.

The UART supports 5, 6, 7, or 8 data bits, plus optional even or odd parity. The transmitters will also supply 1, 2, or fractional stop bits per character and can provide a break output at any time. Summarized as follows are supported modes:

- HDLC with synchronous and asynchronous framing
- Asynchronous and synchronous serial (byte-oriented, like SL-508), parity: none/even/odd/mark/space
- Raw mode (user-controlled FIFO-based synchronous

The DNx-SL-504 are compatible with RS-422 point to point or RS-485 network applications. The ports are based on the Exar SP506CM-L series drivers and provide a wide variety of I/O configurations.

The DNx-SL-504 is supported by the UEIDAQ Framework providing a simple and complete software interface to all popular Windows programming languages and data acquisition/control application packages such as LabVIEW, MATLAB/Simulink, DASYLAB, or any application that supports ActiveX or OPC servers. Support is also provided for all popular “non-Windows” operating systems including Linux, VXworks, QNX, RTX, INtime and more.

As with all UEI PowerDNA boards, the DNx-SL-504 can be operated in harsh environments and has been tested at 5*g* vibration, 50*g* shock, -40 to +85°C temperature, and altitudes up to 70,000 feet. Each board provides 350 V_{rms} Isolation between channels and also between the board and its enclosure or any other installed boards as well as electro-shock-discharge (ESD) isolation.

© Copyright 2012
United Electronic Industries, Inc.

1.3 Features

The common features of the DNx-SL-504 are listed below:

- Four (4) independent serial communication ports
- Each port software-configurable as RS-232 or RS-422/423/485
- Completely independent bit rate settings for every port
- 350V isolation between ports, ports and circuitry; 15kV ESD
- Compatible with RS-422 networks when used in RS-485 mode
- Half- and full-duplex support for RS-485
- Supports HDLC, SDLC, and asynchronous Protocols
- Tested to withstand 5g Vibration, 50g Shock, -40 to +85°C Temperature, and Altitude up to 70,000 ft or 21'000 meters.
- Weight of 136 g or 4.79 oz for DNA-SL-504; 817 g or 28.8 oz with PPC5.
- UEI Framework Software API may be used with all popular Windows programming languages and most real time operating systems such as RT Linux, RTX, or QNX and graphical applications such as LabVIEW, MATLAB, DASYLab and any application supporting ActiveX or OPC.

1.4 Indicators

A photo of the SL-504 unit is illustrated below.

The front panel has two LED indicators:

- RDY: indicates that the layer is receiving power and operational.
- STS: can be set by the user using the low-level framework.

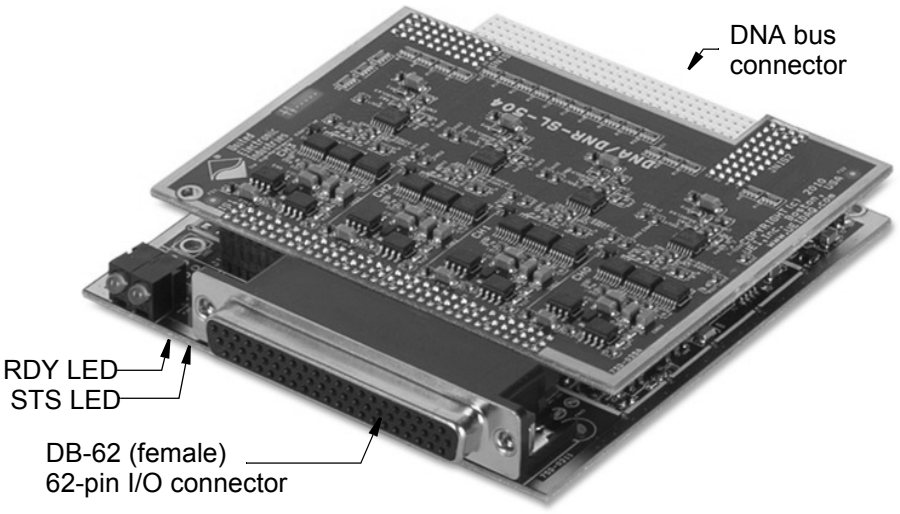


Figure 1-1. The DNA-SL-504 Serial Layer

© Copyright 2012
United Electronic Industries, Inc.

1.5 Specification

The technical specification for the SL-504 is provided in the table below:

Table 1-1. DNx-SL-504 Technical Specifications

Port Specifications	
Number ports	4, independently configurable
UART type	Zilog Z16C32
Interface types	RS-232, RS-422/423, RS-485
Protocols	Async, HDLC, SDLC
FIFOs	32byte, input and output (per port)
Baud rate generator	Programmable, 9.6 kbaud to 4 Mbaud
RS-232 specifications	
RS-232 Asynchronous	256 kbaud
RS-232 Synchronous	600 kbaud
RS-232 Signals	Tx, TxCLK Out, Rx, RxCLK In, CTS, Sync, DCD
RS-485/422 specifications	
RS-485/422 Async	1 Mbaud
RS-485/422 Synchronous	4 Mbaud
RS-485/422 Signals	Tx+, Tx-, TxCLK+, TxCLK-, RX+, RX-, RxCLK+, RxCLK- CTS+. CTS-, DCD+, DCD-
General Specifications	
Isolation	350 V port to port;
ESD protection	15 kV
Power Consumption	2-5W (RS-485 mode with max current drive)
Operating Temperature	Tested -40 to +85 °C
Operating Humidity	0 - 95%, non-condensing
Vibration *IEC 60068-2-6* *IEC 60068-2-64*	5 g, 10-500 Hz, sinusoidal 5 g (rms), 10-500 Hz, broad-band random
Shock *IEC 60068-2-27*	50 g, 3 ms half sine, 18 shocks @ 6 orientations 30 g, 11 ms half sine, 18 shocks @ 6 orientations
MTBF	290,000 hours

© Copyright 2012
United Electronic Industries, Inc.

1.6 What is Serial Communication?

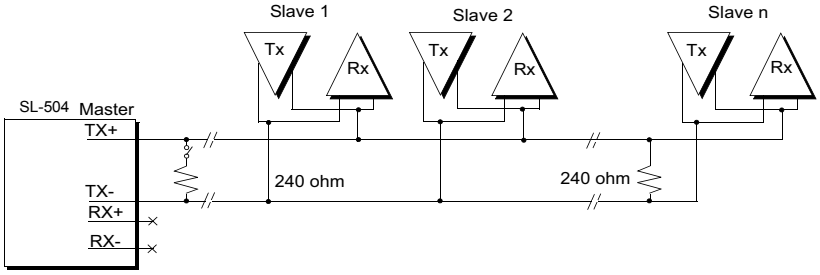
A serial port sends and receives data one bit at a time over one line (composed of a send, a receive, and one common ground wire).

RS-232 is a standard for serial binary data interconnection between a DTE (data terminal equipment) and a DCE (data communication equipment) and normally operates in a bipolar range of –10V to +10V. RS- is an abbreviation for “Recommended Standard”.

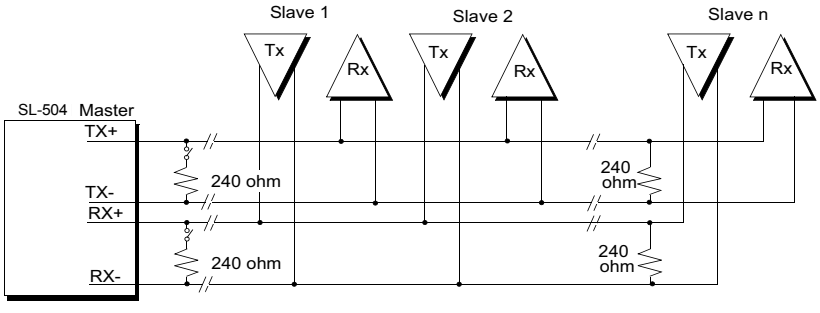
RS-485 (a.k.a EIA-485) is a physical layer electrical specification of a two-wire, half-duplex, multipoint serial connection. A full duplex RS-485 system can be constructed by using two twisted-pair connections (transmit/receive pairs) together as shown in **Figure 1-2**. UART data frames for RS-232 and RS-485 are shown in **Figure 1-3**.

RS-422 is a standard specification for serial communication for information sent bit by bit on a single transmission line, as in RS-232. The difference between the two is that RS-422 is a *balanced differential* signal. RS-422 signal lines consist of two wires, preferably twisted, to reduce noise. The voltage difference difference between the two lines represents the signal value, rather than the voltage level alone. This eliminates a lot of noise induced by external sources and therefore permits higher data rates and cable lengths than RS-232.

There are more differences between RS-422 and RS-232 than maximum data speed and cable length. RS-232 was specifically defined as an interface between computers, printers, and terminals with modems. Furthermore, where the RS-232 line driver is only designed to serve one receiver, a RS-422 line driver can serve up to 10 receivers in parallel. This allows one central control unit to send commands in parallel to as manay as ten slave devices. Unfortunately, those slave devices cannot send information back over a shared interface line. RS-422 allows a multi-drop network topology rather than a multipoint network where all nodes are sonsidered equal and all nodes have send and receive capabilities over the same line. If you need multi-point communication rather than multi-drop, use RS-485 with a maximum of 32 parallel send and 32 receive units parallel on one communication channel.



Two-wire Twisted-Pair Half Duplex Network with 240 ohm Terminating Resistors



Four-wire Twisted-Pair Full Duplex Network with 240 ohm Terminating Resistors

Figure 1-2. RS-485 Topologies

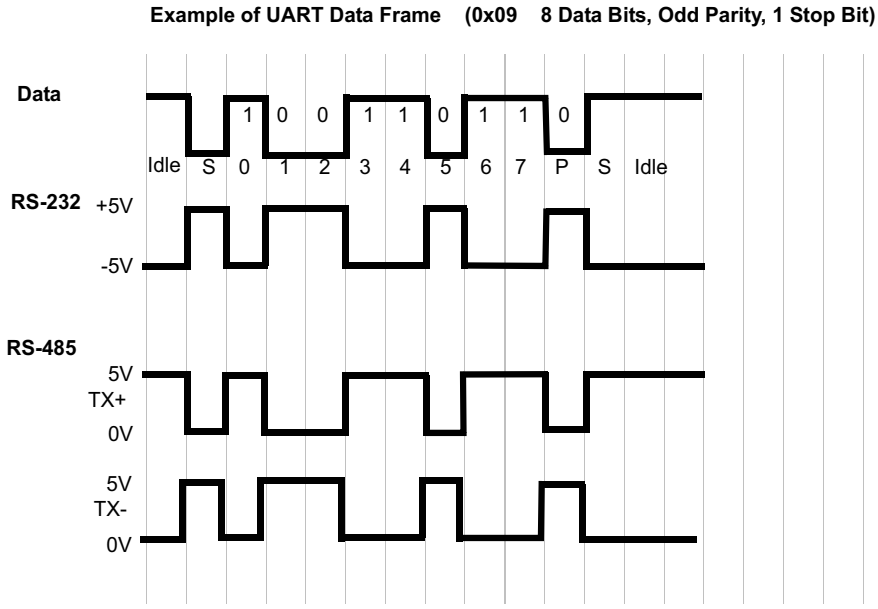


Figure 1-3. UART Data Frames for RS-232 and RS-485

1.7 OSI Layer 1 and Layer 2 Coordination

Referring to the OSI 7-layer model for network communication, Layer 1 is the Physical Link layer or hardware connection between sender and receiver. Layer 2 is the Data Link Layer, the next higher layer in the 7-layer hierarchy. In the UEI SL-504 Synchronous Serial Communication module(s), the physical layer is implemented as an RS-232, RS-422, or RS-485 hardware link. Software-selection of the specific serial line topology is handled on the fly in the 504 by a single chip, the EXAR SP506 Multi-mode Serial Tranceiver. The higher-level synchronous communication functions, associated with Layer 2, the Data Link Layer, are handled by high level data protocols such as SDLC and HDLC running over the physical connections.

© Copyright 2012
United Electronic Industries, Inc.

1.8 What is Synchronous Serial Communication?

Synchronous Serial Communication, as the name implies, is bit- or byte-oriented serial communication in which timing is controlled by an internal or external clock. Referring to the OSI 7-layer model of a network, synchronous serial communications is classed as a Level 2 Data Link Layer. Its function is to transfer data between network entities and also to correct errors that may occur in the Physical Layer (Level 1) below.

SDLC Protocol

The first data link layer protocol based on synchronous, bit-oriented operation, called Synchronous Data Link Control (SDLC), was developed by IBM in the mid-1970s for use in its Systems Network Architecture, in which it was widely used. IBM later submitted SDLC to various standards committees, which eventually resulted in development of a modified SDLC called High-Level Data Link Control (HDLC) protocol. HDLC became the basis for development of several other protocols adopted as standards by the ISO, CCITT, IEEE and other agencies. SDLC, however, remains the primary link layer protocol for SNA environments and WAN links even today.

SDLC can be used with a variety of link types and systems, such as point-to-point and multipoint links, half-duplex and full-duplex transmission, and circuit-switched and packet-switched networks.

SDLC identified two types of network nodes — primary and secondary. Primary nodes control operation of secondary nodes, polling them in a predetermined order. When polled, a secondary can transmit if it has outgoing data to send. The primary node sets up and tears down links and manages each while it is operational. Secondaries can transmit only if the primary grants permission.

The SDLC frame has 6 fields:

- **Flag** – Initiates and terminates error checking.
- **Address** – Contains the SDLC address of the secondary station (source of the frame). A primary is either a communication source or a destination, eliminating the need to include the primary address.
- **Control**
  - **Information Frame (I)** -Carries upper layer and control information. Sends and receives frame sequence numbers; the poll/final (P/F) bit performs flow and error control. The send sequence number defines the number of the next frame to be sent. The receive sequence number provides the number of the frame to be received next. The P/F bit tells the primary station whether or not it requires an immediate response. It is also used to tell a primary that the current frame is the last one in the current response.
  - **Supervisory Frame (S)** – An S frame can request and suspend transmission, report on status, and acknowledge receipt of I frames. S frames do not have information fields.
  - **Unnumbered Frame (U)** – Used to initialize secondaries. Some U frames have an information field.
- **Data –** Contains path or ID information
- **Frame Check Sequence (FCS)** – This is a CRC calculation remainder, used to detect errors.

© Copyright 2012
United Electronic Industries, Inc.

HDLC Protocol

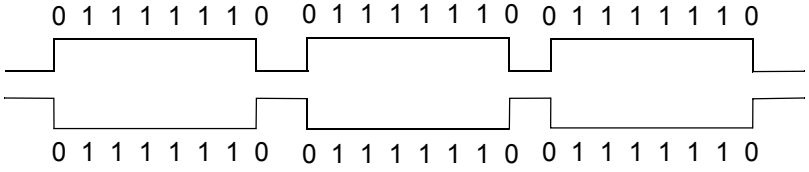
HDLC, the current ISO standard synchronous data link layer protocol, supports three types of transfer modes, whereas SDLC supports only one (Normal Response Mode - NRM). The three modes supported by HDLC are:

- Normal response mode (NRM) – secondaries cannot communicate with a primary until the primary has given permission.
- Asynchronous response mode (ARM) – Secondaries can initiate communication with a primary without receiving permission.
- Asynchronous Balanced Mode (ABM) – In this mode. the node can act as a primary or secondary, depending on the situation. All communication occurs between multiple combined nodes. Any combined station can initiate data transmission without permission from any other.

Framing

HDLC frames can be transmitted over synchronous or asynchronous links. As they have no means for marking the beginning or end of a frame, these points are identified by a frame delimiter, or flag, which is a unique sequence of bits guaranteed not to be seen inside a frame. The sequence is ‘01111110’ (0x7E hex). Each frame begins and ends with a frame delimiter. A frame delimiter at the end of a frame may also mark the start of the next frame. A sequence of 7 or more consecutive 1-bits within a frame causes the frame to be aborted.

When no frames are being transmitted over a simplex or full-duplex synchronous link, a frame delimiter is continuously transmitted on the link. Using the standard NRZI encoding from bits to line levels, this generates one of two continuous waveforms, depending on the initial state, as shown below:



This is used by modems to synchronize their clocks using phase-locked loops. Some protocols allow the “0” bit at the end of a frame to be used as the start of the next frame’s delimiter. For half-duplex or multi-drop communication, a receiver will see continuous idling 1-bits in the inter-frame period when no transmitter is active.

To avoid detecting a false frame delimiter, the receiver must recognize that it has occurred and restore the original data stream before the information is passed to higher layer protocols. On synchronous links, this is accomplished with “bit-stuffing”.

Asynchronous Framing

When using asynchronous serial communication as in standard RS-232 ports, bits are sent in groups of 8 and bit stuffing becomes inconvenient. In such situations, byte or octet-stuffing is used instead, inserting a sequence for 7E (hex) or 7D (hex) when required. If either appears, an escape octet is sent followed by the original data octet with bit 5 inverted. Asynchronous framing is not supported in the current version.

© Copyright 2012
United Electronic Industries, Inc.

Structure

The contents of an HDLC frame are shown below:

Flag	Address	Control	Information	FCS	Flag
8 bits	8 or more bits	8 or 16 bits	Variable length, 0 or more bits	16 or 32 bits	8 bits

Note that the end flag of one frame *may* be the beginning flag of the next frame. Data is usually sent in multiples of 8 (but not in all cases). The FCS is a 16-bit CRC or 32-bit CRC computed over the Address, Control, and Information fields. Its purpose is to detect errors introduced during transmission of the frame. Since the probability of an undetected error occurring increases with data length, the FCS implicitly limits the practical size of the frame. If the receiver-calculated FCS does not match that of the sender, the receiver either sends a negative acknowledge packet or sends nothing. The sender can then retransmit the failed frame.

SDLC was originally designed to connect one computer with multiple peripherals in a master/slave relationship in which the computer (primary terminal) gives each peripheral (secondary terminal) permission to speak in turn. Only one address, that of the secondary terminal, is used. Commands and responses are the same -- they only difference between them is the direction of transmission.

In HDLC, normal response mode allows operation over half-duplex links, provided the primary station is aware that it cannot transmit when it has given permission to a secondary. Asynchronous response mode is an HDLC addition for use with full-duplex links. It allows a secondary station to transmit at any time. Asynchronous balance mode added the concept of a combined terminal (both primary and secondary)

There are three basic types of HDLC frames: (I) Information Frames, (S) Supervisory Frames, and (U) Unnumbered Frames,

I-frames transport user data from the network layer, combined with flow and error control information.

S-frames do not have information fields, but are used for flow and error control when no data is sent.

U-frames are used for miscellaneous purposes, such as link management., and may contain an information field.

P/F Bit

Poll/Final is a single bit with two names: **Poll** when set by the primary station to get a response from a secondary, and **Final** when set by a secondary station to indicate a response or the end of transmission. Otherwise, it is clear.

The bit is used as a token, passed back and forth between stations. Only one can exist at a time. The secondary only sends a Final when it has received a poll from the primary. The primary only sends a poll when it has received a Final from the secondary, or after a timeout indicating a lost bit.

In normal response mode (NRM), possession of the token also grants the secondary permission to transmit. The secondary sets the F-bit in its last response frame and gives up the token.

In the other modes, ARM and ABM, the P bit forces a response.

If no response to a P bit is received in a reasonable time, the primary times out and sends P again.

© Copyright 2012
United Electronic Industries, Inc.

The P/F bit is the heart of the basic checkpoint retransmission scheme required by HDLC. Whenever a station receives a P/F bit, it can assume that any frames sent before it transmitted the P/F bit and not acknowledged, will never arrive, and so should be retransmitted.

N(R)

Both I and S frames contain a receive sequence number, N(R). It provides a positive acknowledgement that N(S) numbers up to N(R)-1 have been received and indicates the N(S) value of the next frame it expects to receive.

I-Frames (user data)

I-frames transport user data from the network layer and also include flow and error control information.

S-Frames (control)

S-frames are primarily used for flow and error control. S-frames do not have information fields. All S-frames include a P/F bit and a receive sequence number. There is no difference between a command S frame and a response S frame. The first 2 bits mean it is an S-frame and identify its type: (RR), (RNR), (REJ), or (SREJ)

(RR) indicates the sender is ready to receive more data.

(RNR) indicates a request not to send more data until further notice.

(REJ) requests immediate retransmission starting with N(R).

(SREJ) requests retransmission of only the frame N(r).

U-Frames

U-frames are used for link management, but can also be used to transfer user data. Codes bit fields indicate the type of U-frame.

For more detailed information about HDLC, its command and response repertoire, basic operations and functional extensions, refer to readily available resources such as wikipedia and similar communication reference material.

Note that the SL-504 provides only a means to send and receive frames, but does not implement control part of HDLC/SDLC protocol (layer 3).

© Copyright 2012
United Electronic Industries, Inc.

1.9 Architecture

The architecture of the DNx-SL-504 is illustrated in the block diagram shown in **Figure 1-4**.

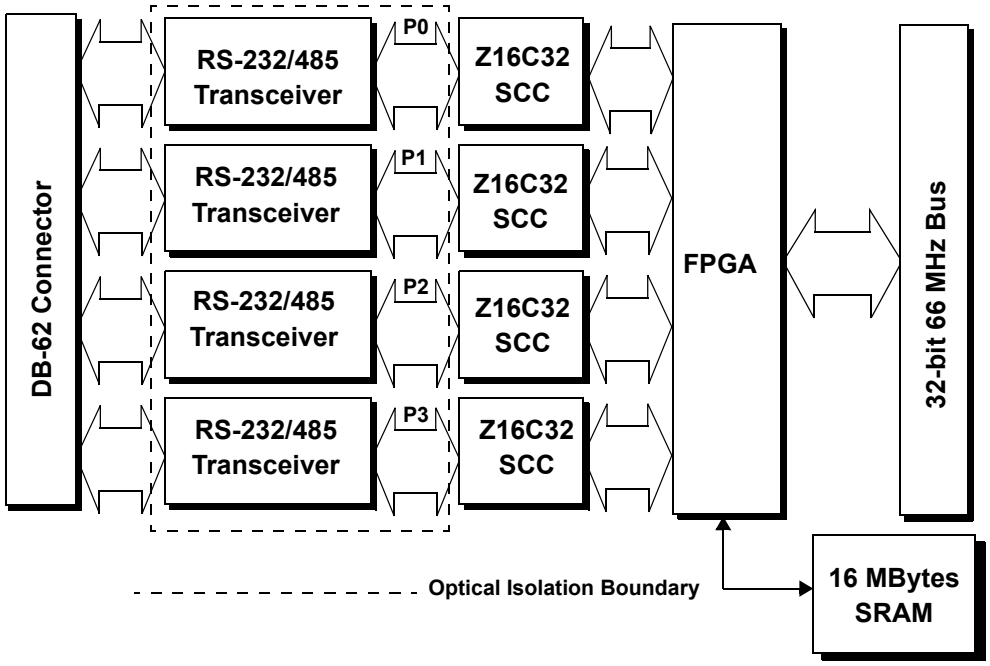


Figure 1-4. Logic Block Diagram: DNA/DNR-SL-504 Overview

The DNx-SL-504 is equipped with four software-controlled EXAR SP506 single chip multi-mode serial line transceivers, one per channel, that select the type of DTE to DCE connectivity to be used for the physical layer communication. The SP506 chip also provides an internal loop-back feature that eliminates the need for a loopback plug. In loopback mode, driver outputs are internally connected to receiver inputs, creating an internal path for diagnostic testing. This chip is an integrated design that also provides any necessary termination hardware that may be needed.

The interface modes supported by the SP506 chip are: RS-232, RS-422, EIA-530, EIA-530A,RS-449, V.35, V.36, and RS-485. It achieves differential transmission rates of more than 20 Mbps. In the SL-504, the modes offered are: RS-232, RS-422, and RS-485.

The `SP506 Transceiver` is controlled by an FPGA, the board control chip. The FPGA works in conjunction with the core module logic of the DNA or DNR chassis.

The SL-504 Module consists of two PC boards, one of which is a 61x Base Board and another, which is the four port 504-specific I/O board.The 504-specific board plugs into a bus connector on the base board.

© Copyright 2012
United Electronic Industries, Inc.

Z16C32 IUSC Integrated Universal Serial Controller

The Z16S32 IUSC Integrated Universal Serial Controller is a multi-protocol data communications device with on-chip dual-channel DMA. Integration of a high-speed serial data comm channel with high performance DMA affords a higher data throughput than can be achieved with discrete components. The Z16C32 chip, manufactured by Zilog is software configurable to satisfy a wide variety of serial communications applications. The high-speed data rate and multi-protocol support make it ideal for today's application and speed requirements. The chip offers many advanced functions in an integrated design such as two baud rate generators, a digital phase-locked loop, character counters, and 32-byte FIFOs for both receiver and transmitter.

The IUSC handles asynchronous formats, and synchronous bit-oriented formats such as HDLC. This device supports virtually any serial data transfer application.

The chip can generate and check CRC in any synchronous mode. Complete access to the CRC value allows system software to resend or manipulate the CRC as needed in various applications.

1.10 Module Capabilities

Using the RS-232 or RS-485 standard, the controller is capable of communicating at speeds of 256Kbaud for RS-232 and 4Mbaud for RS-485. When in RS-485 mode, the SL-504 is also compatible with RS-422 networks.

Each port has independently programmable:

- Baud/bit rate
- HDLC format
- Short flag
- Idle character
- Encoding
- Clocking scheme
- CRC
- Filtering Type
- Preamble size and type

© Copyright 2012
United Electronic Industries, Inc.

1.11 Wiring & Connectors

The following signals are located at the DB-62 connector on the SL-504 module:

Signals on DB-62 Connector

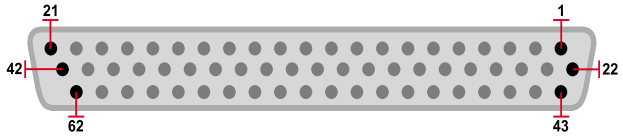
The 62-pin female D-Sub connector on the SL-504 module is divided into four 9-pin serial ports as shown in the pinout of **Figure 1-5**. A user can connect four serial lines to this connector either through a custom made cable or by connecting to a DNA / DNR-STP-62 accessory panel (see Appendix, page 35).

In the pinout of **Figure 1-5** the following abbreviations are used:

Name	+/- 232	+/- 422/485	Description (RS-422/485)
TxD/RxD	single-ended	a,b/a,b	Transmit/Receive Data pair
“CTS”	single-ended		Clear-to-send (input)
“DCD”	single-ended		Data Carrier Detect (input)
TxC/ RxC	single-ended/ single-ended	a,b a,b	Transmit Clock (output)/ Receive Clock (input)
GND		single-ended	Common Ground

Table 1-2. Abbreviations for pinout; with (a) high +, (b) low -

The pinout for the SL-504 is provided below:



Pin	signal	Pin	signal	Pin	signal
1	RESERVED01	22	GND-CH0	43	RxC(a)-0
2	TxC(b)-0	23	TxC(a)-0	44	RxC(b)-0
3	DCD(a)-0	24	DCD(b)-0	45	GND-CH0
4	RxD(b)-0	25	RxD(a)-0	46	RESERVED11
5	TxD(b)-0	26	TxD(a)-0	47	CTS(b)-0
6	GND-CH1	27	CTS(a)-0	48	RESERVED17
7	RxC(b)-1	28	RxC(a)-1	49	DCD(b)-1
8	TxC(b)-1	29	TxC(a)-1	50	DCD(a)-1
9	RxD(b)-1	30	RxD(a)-1	51	GND-CH1
10	TxD(b)-1	31	TxD(a)-1	52	CTS(a)-1
11	RESERVED33	32	CTS(b)-1	53	RxC(a)-2
12	TxC(b)-2	33	TxC(a)-2	54	RxC(b)-2
13	DCD(b)-2	34	GND-CH2	55	GND-CH2
14	RxD(b)-2	35	RxD(a)-2	56	DCD(a)-2
15	TxD(b)-2	36	TxD(a)-2	57	CTS(b)-2
16	GND-CH3	37	CTS(a)-2	58	RESERVED49
17	RxC(b)-3	38	RxC(a)-3	59	DCD(b)-3
18	TxC(b)-3	39	TxC(a)-3	60	RESERVED55
19	DCD(a)-3	40	GND-CH3	61	RxD(a)-3
20	TxD(b)-3	41	TxD(a)-3	62	RxD(b)-3
21	CTS(a)-3	42	CTS(b)-3		

Figure 1-5. DNx-SL-504 Connection Diagram

© Copyright 2012
United Electronic Industries, Inc.

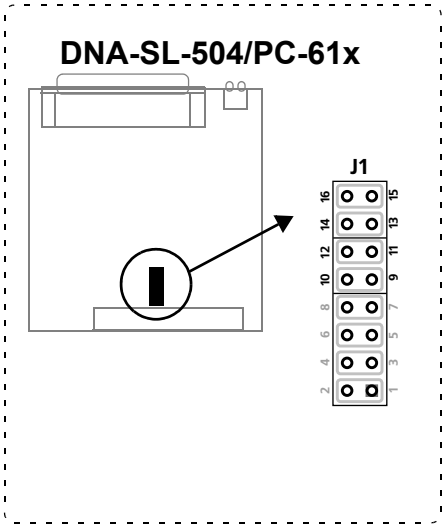
1.12 Jumper Settings for Board Position

This section briefly describes how to change jumper positions that indicate logical layer positioning in the uncommon case that you must physically swap layers on the PowerDNA I/O Cube (not PowerDNR RACKtangle).

The base board of a DNA-SL-504 module (layer) has a jumper block that assigns the position of the module within a PowerDNA Cube. The jumpers must be set to match the physical position of an I/O board or layer in the Cube. Board position jumpers are not provided with the DNR versions of the SL-504. The physical position of the board within the DNR RACKTangle™ enclosure is determined automatically by the system.

All layers are assembled in Cubes before shipment with identifying labels, so you should never have to change a jumper setting unless you have to change a layer from one physical position to another in the field.

Figure 1-6 shows the physical layout of DNA-SL-504 Board, highlighted to show the 16-pin jumper block for setting the board position within a PowerDNA Cube.



See fig. 1-7 for placement of jumpers for various board positions in a Cube.

Figure 1-6. Jumper Block for DNA-SL-504 Board Position

A diagram of the jumper block is shown in **Figure 1-7**. To set the board position jumpers, place jumpers as shown in **Figure 1-7**.

		Layer's Position as marked on the Faceplate*					
		I/O **1**	I/O **2**	I/O **3**	I/O **4**	I/O **5**	I/O **6**
Jx Pins	9-10	■	□	■	□	■	□
	11-12	■	■	□	□	■	■
	13-14	■	■	■	■	□	□
	15-16	■	■	■	■	■	■

* *All I/O Layers are sequentially enumerated from top to the bottom of the Cube*

□ - **Open** ■ - **Closed**

Figure 1-7. Diagram of DNA-SL-504 Board Position Jumper Settings

© Copyright 2012
United Electronic Industries, Inc.

Chapter 2 Programming with the High Level API

This section describes how to control the DNx-SL-504 using the UeiDaq Framework High Level API.

UeiDaq Framework is object oriented and its objects can be manipulated in the same manner from different development environments such as Visual C++, Visual Basic or LabVIEW.

The following section focuses on the C++ API, but the concept is the same no matter what programming language you use.

Please refer to the “UeiDaq Framework User Manual” for more information on use of other programming languages.

2.1 Creating a Session

The Session object controls all operations on your PowerDNx device. Therefore, the first task is to create a session object:

```
// create a session object for input, and a session object for output
CUeiSession slSession;
```

2.2 Configuring the Resource String

UeiDaq Framework uses resource strings to select which device, subsystem and channels to use within a session. The resource string syntax is similar to a web URL:

```
<device class>://<IP address>/<Device Id>/<Subsystem><Channel list>
```

For PowerDNA and RACKtangle, the device class is **pdna**.

For example, the following resource string selects serial ports 0,2,3 on device 1 at IP address 192.168.100.2: “pdna://192.168.100.2/Dev1/Com0,2,3”

The SL-504 is programmed using the subsystem **com** to configure channels in asynchronous mode or **hdlc** to configure channels in HDLC mode.

2.2.1 Asyncronous serial port

Use the method **CreateSerialPort()** to configure one or more channel(s) in asynchronous mode.

The following call configures the ports 0 and 1 of an SL-504 set as device 1:

```
// Configure session’s ports
slSession.CreateSerialPort("pdna://192.168.100.2/Dev1/Com0,1",
                           UeiSerialModeRS232,
                           UeiSerialBitsPerSecond57600,
                           UeiSerialDataBits8,
                           UeiSerialParityNone,
                           UeiSerialStopBits1,
                           termination);
```

© Copyright 2012
United Electronic Industries, Inc.

The string configures the following parameters:

- **Serial port mode**: The physical interface used to transmit serial bytes:
  UeiSerialModeRS232: RS-232
  UeiSerialModeRS485HalfDuplex: RS-422 (2 wires)
  UeiSerialModeRS485FullDuplex: RS-485 (4 wires)
- **Bits per second**: The number of bits per second to transmit. To set a custom baud for a serial port, obtain a reference the session's channel, then set the configuration in the relevant section. For example, in C#:
```
UeiDaq.SerialPort p=(SerialPort)session.GetChannel(0);
p.SetCustomSpeed(230400); // set to 230400 baud
```
- **Data bits**: The number of data bits per serial byte:
  UeiSerialDataBits5: 5 bits of data
  UeiSerialDataBits6: 6 bits of data
  UeiSerialDataBits7: 7 bits of data
  UeiSerialDataBits8: 8 bits of data
- **Parity**: The parity used to verify data integrity:
  UeiSerialParityNone: No parity bit
  UeiSerialParityOdd: parity bit set to 1 if number of bits set to 1 is odd
  UeiSerialParityEven: parity bit set to 1 if number of bits set to 1 is even
  UeiSerialParityMark: parity bit is always 1
  UeiSerialParitySpace: parity bit is always 0
- **Stop bits**: The number of stop bits sent at the end of each character:
  UeiSerialStopBits1: 1 stop bit
  UeiSerialStopBits1_5: 1.5 stop bits
  UeiSerialStopBits2: 2 stop bits
- **Termination**: termination string that marks the end of transmission (not used on SL-504)

In addition you can set additional parameters using the channel object methods (under LabVIEW use property node):

- **Termination resistors**: Select whether to enable or disable termination resistors.
```
// enable termination resistor
pPort->EnableRxTerminationResistor(true);
```
- **Echo suppression**: Select whether to suppress echo in half duplex mode
```
// Disable echo
pPort->EnableHDEchoSuppression(true);
```
- **Error reporting**: Select whether to report errors such as framing or parity errors.
```
// Enable error reporting
pPort->EnableErrorReporting(true)
```

© Copyright 2012
United Electronic Industries, Inc.

2.2.2 HDLC Port

Use the method **CreateHDLCPort()** to to configure one or more channel(s) in asynchronous mode.

The following call configures ports 2 and 3 of a SL-504 set as device 1:

```
// Configure session's ports

hdlcSession.CreateHDLCPort("pdna://192.168.100.2/Dev1/hdlc2,3",
                           UeiHDLCPortRS232,
                           100000,
                           UeiHDLCPortEncodingNRZ,
                           UeiHDLCPortCRCNone,
                           UeiHDLCPortClockBRG,
                           UeiHDLCPortClockExternal);
```

It configures the following parameters:

- **Physical interface**: the physical interface used to transmit serial bytes.
  UeiHDLCPortRS232: RS-232
  UeiHDLCPortRS422: RS-422 (2 wires)
  UeiHDLCPortRS485: RS-485 (4 wires)
  UeiHDLCPortV35: V35
- **Bits per second**: the number of bits per second transmitted of the synchronous port
- **Encoding**: the method used to encode bits over synchronous serial line:
  UeiHDLCPortEncodingNRZ: NRZ encoding
  UeiHDLCPortEncodingNRZB: inverted NRZ encoding
  UeiHDLCPortEncodingNRZI: NRZI encoding
  UeiHDLCPortEncodingNRZIMark: NRZI encoding, invert state for 1
  UeiHDLCPortEncodingNRZISpace: NRZI encoding, invert state for 0
  UeiHDLCPortEncodingBiphaseMark: biphase encoding, with DPLL
  UeiHDLCPortEncodingBiphaseSpace: biphase encoding, with DPLL
  UeiHDLCPortEncodingBiphaseLevel: biphase encoding, with DPLL
  UeiHDLCPortEncodingBiphaseDiff: biphase encoding, used with DPLL
- **CRC**: the method used to calculate the cyclic redundancy code:
  UeiHDLCPortCRCNone: CRC is not checked neither for transmit nor receive
  UeiHDLCPortCRCUser: User responsible for inserting & checking CRC
  UeiHDLCPortCRC16CCITT: use 16-bit CCITT CRC (x^15+x^12+x^5+1)
  UeiHDLCPortCRC16: 16-bit polynomial
  UeiHDLCPortCRC32:-32-bit polynomial
- **TX clock source**: clock source used to synchronize transmitter:
  UeiHDLCPortClockExternalPin: Take clock from TxC pin (default)
  UeiHDLCPortClockBRG: Take clock from baud rate generator
  UeiHDLCPortClockDPLL: Take clock from DPLL divided by 32
  UeiHDLCPortClockDPLLDiv8: Take clock from DPLL divided by 8
  UeiHDLCPortClockDPLLDiv16: Take clock from DPLL divided by 16
- **RX clock source**: clock source used to synchronize receiver:
  UeiHDLCPortClockExternalPin: Take clock from RxC pin (default)
  UeiHDLCPortClockBRG: Take clock from baud rate generator
  UeiHDLCPortClockDPLL: Take clock from DPLL divided by 32
  UeiHDLCPortClockDPLLDiv8: Take clock from DPLL divided by 8
  UeiHDLCPortClockDPLLDiv16: Take clock from DPLL divided by 16

© Copyright 2012
United Electronic Industries, Inc.

In addition you can set the following parameter using the channel object methods (under LabVIEW use property node):

- **Termination**: Select whether to enable or disable termination resistors.
```
// enable termination resistor
pPort->EnableTerminationResistor(true);
```
- **Echo suppression**: Select whether to suppress echo in half duplex mode (RS-422)
```
// Disable echo
pPort->EnableHDEchoSuppression(true);
```
- **Loopback**: Select whether transmitter and receiver of the same port are tied.
```
// Disable loopback
pPort->EnableLoopback(false);
```
- **Abort Symbol**: The symbol used to abort
UeiHDLCPortAbort7: Send 0x7F to abort
UeiHDLCPortAbort15: Send 0x7FFF to abort
```
// Set abort symbol to 0x7F
pPort->SetAbortSymbol(UeiHDLCPortAbort7);
```
- **Underrun Action**: The action taken when underrun condition is detected:
UeiHDLCPortUnderrunFinish: Close the frame by adding CRC to it
UeiHDLCPortUnderrunFlags: Send flags
```
// Set underrun action
pPort->SetUnderrunAction(UeiHDLCPortUnderrunFinish);
```
- **Filter Mode**: The filter setting:
UeiHDLCPortFilterNone: No filtering
UeiHDLCPortFilterA16: +16 bits into RxFIFO if Addr matches or B/C as 2 bytes
UeiHDLCPortFilterA24: +24 bits into RxFIFO if Addr matches or B/C as 3 bytes
UeiHDLCPortFilterA32: +32 bits into RxFIFO if Addr matches or B/C as 4 bytes
UeiHDLCPortFilterEALS: Places bytes while LS==0, then byte with LS==1 then 16 bits as 2 bytes into RxFIFO if EA matches or B/C
UeiHDLCPortFilterEA24: Places 24 bits as 3 bytes into RxFIFO if EA matches or B/C
UeiHDLCPortFilterEAMS: Places bytes while MS==0, then byte with MS==1 then 8 bits as 1 byte into RxFIFO if Ext Addr matches or B/C
UeiHDLCPortFilterEAMS16: Places bytes while MS==0, then byte with MS==1 then 16 bits as 2 bytes into RxFIFO if Ext Addr matches or B/C
```
// Disable filter
pPort->SetFilterMode(UeiHDLCPortFilterNone);
```
- **Filter address**: The address to filter
```
// Set address to filter
pPort->SetFilterAddress(0x52);
```
- **Idle flags**: The pattern to transmit when the link is idle.
UeiHDLCPortIdleFlag: continuous flags
UeiHDLCPortIdleZero: continuous zeroes
UeiHDLCPortIdleOne: continuous ones
UeiHDLCPortIdleMark: idle chars are marks
UeiHDLCPortIdleSpace: idle chars are spaces

© Copyright 2012
United Electronic Industries, Inc.

UeiHDLCPortIdleMS: alternating Mark and Space
UeiHDLCPortIdle01: .alternating 0 and 1

```
// Set idle pattern
pPort->SetidleCharacter(UeiHDLCPortIdleOne);
```

2.3 Configuring the Timing

The application must configure the SL-504 to use the “messaging” timing mode.

A message is represented by an array of bytes.

The SL-504 can be programmed to wait for a certain number of bytes to be received before notifying the session.

It is also possible to program the maximum amount of time to wait for the specified number of bytes before notifying the session.

The following sample shows how to configure the messaging I/O mode to be notified when 10 bytes have been received or every second, whichever is less. (Note that if the serial port receives less than 10 bytes per second, it will return whatever number of bytes are available every second).

```
// configure timing of serial port

session.ConfigureTimingForMessagingIO(10, 1.0);
```

2.4 Read Data

Reading data from the SL-504 is done using a *reader* object. As there is no multiplexing of data (contrary to what’s being done with AI, DI, or CI sessions), you need to create one reader object per serial port to be able to read from each port in the port list.

2.4.1 Asynchronous Mode

The following sample code shows how to create a reader object tied to port 1 and read up to 10 bytes from the serial port.

```
// Create a reader and link it to the session’s stream, port 1

reader = new CUeiSerialReader(slSession.GetDataStream(), 1);

// we’ll want to store for 10 bytes (char-sized)

Unsigned char bytes[10];

// read up to 10 bytes, numBytesRead contains
// the number of bytes actually received

reader->Read(10, bytes, &numBytesRead);
```

2.4.2 HDLC Mode

The following sample code shows how to create a reader object tied to port 1 and read up to 10 bytes from that HDLC port.

```
// Create a reader and link it to the session’s stream, port 1

reader = new CUeiHDLCReader(hdlcSession.GetDataStream(), 1);

// we’ll want to store for 10 bytes (char-sized)

uInt8 bytes[10];

// read up to 10 bytes

reader->Read(10, bytes, &numBytesRead);
```

© Copyright 2012
United Electronic Industries, Inc.

2.5 Write Data

Writing data to the SL-504 is done using a *writer* object. As there is no multiplexing of data (contrary to what's being done with AO, DO, or CO sessions), you need to create one writer object per serial port to be able to write to each port in the port list.

2.5.1 Asynchronous Mode

The following sample code shows how to create a writer object tied to port 2 and send one byte to the serial port.

```
// Create a writer and link it to the session's stream, port 2
writer = new CUeiSerialWriter(session.GetDataStream(), 2);
// store the one byte we want to write out
unsigned char bytes[2] = {0x23, 0};
// write 1 byte, numBytesWritten contains number of bytes actually sent
writer->Write(1, bytes, &numBytesWritten);
```

2.5.2 HDLC Mode

The following sample code shows how to create a writer object tied to port 2 and send a frame of 128 bytes to the HDLC port.

```
// Create a writer and link it to the session's stream, port 2
writer = new CUeiSerialWriter(session.GetDataStream(), 2);
// store 128 bytes that we want to write out
unsigned char bytes[128];
memset(bytes,0x34,128);
// write 128 byte, numBytesWritten contains number of bytes actually sent
writer->Write(128, bytes, &numBytesWritten);
```

2.6 Cleaning-up the Session

The session object will clean itself up when it goes out of scope or when it is destroyed. To reuse the object with a different set of channels or parameters, you can manually clean up the session as follows:

```
// clean up the sessions
slSession.CleanUp();
```

© Copyright 2012
United Electronic Industries, Inc.

Chapter 3 Programming with the Low-level API

This chapter illustrates how to program the PowerDNA cube using the low-level API. The low-level API offers direct access to PowerDNA DAQBios protocol and also allows you to access device registers directly.

However, we recommend that, when possible, you use the UeiDaq Framework High-Level API, because it is easier to use. You should need to use the low-level API only if you are using an operating system other than Windows.

For additional information about low-level programming of the SL-504, please refer to the PowerDNA API Reference Manual document under:

Start » Programs » UEI » PowerDNA » Documentation

3.1 HDLC/SDLC Mode

The HDLC protocol was developed in 1970s to facilitate synchronous transmission of data packets (or frames) over the RS-485 or RS-232 physical interface. Since this is a synchronous protocol, either clock & data lines or an encoding with embedded clock can be used (effectively cutting bandwidth by half for the same baud rate).

When an HDLC transmitter is enabled, it constantly sends Idle characters on the bus. When a transmitter needs to start transmission, it sends a Flag sequence (0x7E). Six continuous “ones” on the bus signals a receiver that this is a start of the frame. A frame is a group of sequential characters ending with CRC to verify its correctness. While sending a frame, an HDLC transmitter continually checks whether any sequence of data bits could look like a Flag to the receiver. It does this without regard for character boundaries. Whenever the data presented to a transmitter includes a “zero” followed by five “ones”, the transmitter adds an extra “zero” bit after a fifth “one” bit. The receiver monitors the serial data stream as well and removes a trailing zero for any sequence that looks like 0111110, regardless of character boundaries.

HDLC protocol matches up well with NRZI-Space encoding to ensure data transmissions for clock resynchronization. This is because the transmitter inverts NRZI-space data for every “zero” bit and there are never more than five “one” bits in succession within a frame.

Since the flag-matching hardware operates without regard for character boundaries, bit oriented synchronous protocols can handle any number of bits in length. The current implementation of the SL-504 card allows the transmission of multiple of 8-bit character and reception of any number of bits. It limits the maximum frame size of both receiver and transmitter to 4096 8-bit characters, not including CRC.

© Copyright 2012
United Electronic Industries, Inc.

3.2 Asynchronous Mode

Asynchronous mode supports both RS-232 and RS-485 protocols. The user can select the number of start and stop bits, parity mode, and number of data bits per each character.

3.2.1 Setting the configuration

DqAdv504SetConfig() is used to set up channel configuration:

```
DqAdv504SetConfig(int handle, int devn, int channel,
SL504_SETCFG* config);
```

SL504_SETCFG is defined as follows:

```
typedef struct {
    uint32 protocol;        // protocol used:
                            // HDLC/sync/async/flags
    uint32 modeflags;       // additional flags for
                            // mode selection
    uint32 physical;        // physical interface
                            // RS-232/422/485
 // fill the following section for HDLC or RawSync modes
    uint32 hdlc_flags;      // additional
                            // synchronous mode flags
    uint32 hdlc_encod;      // encoding type - NRZ,
                            // MII, etc.
    uint32 hdlc_baud;       // RxC or TxC clock rate
                            //in baud
    uint32 hdlc_clk_src;    // Which clock to
                            // synchronize Tx from
    uint32 hdlc_crc_mode;   // None, CRC-16/32/
            // CCITT, user-supplied or automatic
    uint32 hdlc_flt_mode;   // Filter type
    uint32 hdlc_filter;     // HDLC address filter
                             // value (8 bits)
    uint32 hdlc_preamble;   // preamble
    uint32 hdlc_prmbl_sz;   // size of preamble
    uint32 hdlc_idle_ch;    // idle character
                            // (default 0x7E)
 // fill the following section for asynchronous mode
    uint32 async_baud;      // baud rate
    uint32 async_char_sz;   // 5 to 8 bits before
                            // parity bit, 9th bit
                            // is after parity
    uint32 async_start;     // number of start bits
                            // (=0 is default means 1 bit)
    uint32 async_stop;      // number of stop bits
                            // (=0 is default means 1 bit)
    uint32 async_parity;    // parity used
    uint32 async_msglen;    // watermark level defines msg len
    uint32 async_tout;      // timeout forces to
                            // transmit everything
                            // accumulated (in ms)
} SL504_SETCFG, *pSL504_SETCFG;
```

This structure is used for all modes of operation albeit only a part of it is valid for each mode.

For HDLC mode <protocol> can be selected from:

```
// <protocol>
#define SL504_PROT_HDLC         1   // HDLC protocol support
#define SL504_PROT_ASYNC        2   // Async mode (char-based)
#define SL504_PROT_SYNC         3   // <reserved>
// Bit-synchronous, external sync, no framing or
// synchronization
#define SL504_PROT_ISOCHRONOUS  4   // <reserved>
                                    // isochronous mode
#define SL504_PROT_NINEBIT      5   // <reserved>
                                    // 9th bit address mode
#define SL504_PROT_RESV0        6   // <reserved>
#define SL504_PROT_BIMONOSYNC   7   // <reserved>
                                    // monosync or bisync mode
#define SL504_PROT_T_BISYNC     8   // <reserved>
                                    // transparent bisync mode
#define SL504_PROT_S_MONOSYNC  10   // <reserved>
                                    // slaved monosync mode
```

Only two protocols are currently implemented:

- SL504_PROT_HDLC – works with HDLC and SDLC protocol
- SL504_PROT_ASYNC - works with asynchronous protocol

Once a protocol is selected, there are a few protocol modifiers for HDLC mode:

<modeflags>:--

```
#define SL504_HDLC_LOOP         (1L<<0)   // HDLC loop mode
#define SL504_USE_CTS           (1L<<4)  // CTS input, this
                                       // controls transmission
#define SL504_USE_DCD           (1L<<5)  // DCD
                                         // enables receiver
#define SL504_INHIBIT_TX        (1L<<6)  // Do not
                                         // enable transmitter
#define SL504_INHIBIT_RX        (1L<<7)  // Do not
                                         // enable receiver
```

The SL-504 layer has two inputs per channel: one is CTS and the second one is DCD. By selecting bits 4 and 5, the user gives control over transmission enable to CTS input and/or for receiver enable to DCD line.

User can also force transmitter and/or receiver into disabled state if appropriate flags are added to the <modeflags> word.

<physical> applies to all modes of operation and selects the physical interface the SL-504 board uses to communicate with other devices.

```
#define SL504_PHY_RS232         1   // RS-232 up to 230k baud
#define SL504_PHY_RS485         2   // RS-485 up to 4Mbit
#define SL504_PHY_RS422         3   // RS-422 multidrop
#define SL504_PHY_V35           4   // balanced
// current data and clock and unbalanced voltage DCD and CTS

#define SL504_PHY_TERM          0x40  // enable termination in
                                    // RS-485 and RS-422 modes
#define SL504_PHY_LOOP          0x80   // enable loopback
                                       // in the selected mode
#define SL504_PHY_NOECHO        0x100  // suppress echo
                                       // in RS-422/423 modes
```

For the above mode definitions,

- <SL504_PHY_TERM> enables termination on RS-485 or RS-422 bus
- <SL504_PHY_LOOP> sets Tx chip into loopback mode so all transmitted data is received by the receiver without a need for an external dongle
- <SL504_PHY_NOECHO> is used to disable the receiver in RS-422 mode while the transmitter transmits data on the bus. Since this feature is implemented in the software, it only works without queuing of transmit packets, thus the depth of the transmit DMA FIFO becomes one frame.

<hdlc_flags> define how to handle HDLC protocol

```
#define SL504_HDLC_ABORT_7        (0L<<0)   // send
// 0x7f as an abort symbol
#define SL504_HDLC_ABORT_15       (1L<<0)   // send
// 0x7fff as an abort symbol
#define SL504_HDLC_FINISH_UNDER   (2L<<0)   // in
// underrun condition close the frame by adding CRC to it
#define SL504_HDLC_FLAGS_UNDER    (3L<<0)   // in
// underrun condition start sending flags
#define SL504_HDLC_SHARED_ZEROES  (1L<<6)   // send
//  idle flags with shared zeroes
```

<hdlc_encod> selects what type of encoding is used. The most popular type of encoding is NRZ, other types of encodings are rarely used.

```
#define SL504_HDLC_NRZ            (0L<<0)   // NRZ
// encoding
#define SL504_HDLC_NRZB           (1L<<0)   //
// inverted NRZ encoding
#define SL504_HDLC_NRZI           (2L<<0)   // NRZI
// encoding
#define SL504_HDLC_NRZI_MARK      (3L<<0)   // NRZI
// encoding, invert state for 1
#define SL504_HDLC_NRZI_SPACE     (4L<<0)   // NRZI
// encoding, invert state for 0
#define SL504_HDLC_BIPHASE_MARK   (5L<<0)   //
// biphase encoding, used with DPLL
#define SL504_HDLC_BIPHASE_SPACE  (6L<<0)   //
// biphase encoding, used with DPLL
#define SL504_HDLC_BIPHASE_LEVEL  (7L<<0)   //
// biphase encoding, used with DPLL
#define SL504_HDLC_BIPHASE_DIFF   (8L<<0)   //
// biphase encoding, used with DPLL
```

NRZ (non-return to zero) encoding requires synchronous clock. Thus both TxD and TxC and/or RxD and RxC pairs need to be used.

"One" is represented by one physical level, "Zero" is represented by another level (usually a positive voltage). Actual “one” and “zero” representation depends on the physical interface selected.

In NRZI Space, "One" is represented by no change in physical level. "Zero" is represented by a change in physical level. In clock language, the level transitions on the trailing clock edge of the previous bit to represent a "zero."

© Copyright 2012
United Electronic Industries, Inc.

3.3 NRZI Mark Encoding

This "change-on-zero" is often used by High-Level Data Link Control. HDLC avoids long periods of no transitions (even when the data contains long sequences of 1 bits) by using zero-bit insertion. HDLC transmitters insert a 0 bit after five contiguous 1 bits (except when transmitting the frame delimiter '01111110'). The receiver automatically detects and deletes inserted zeros.

NRZI Inverted is another form of the NRZ encoding. The two level NRZI signal has a transition at a clock boundary if the bit being transmitted is a logical 1, and does not have a transition if the bit being transmitted is a logical 0. "One" is represented by a transition of the physical level. "Zero" has no transition.

Biphase encoding does not require a synchronous clock to be used since the clock is embedded into the data signal. Biphase encoding is a code in which data and clock signals are combined to form a single 2-level self-synchronizing data stream. It is a differential encoding, using the presence or absence of transitions to indicate logical value. A transition is guaranteed at least once every bit, allowing the receiving device to perform clock recovery. The Z16C32 chip uses an integrated digital phase lock loop (DPLL) to recover clock and uses it instead of external clock.

The symbol rate is twice the bit rate of the NRZ signal. Each bit period is divided into two half-periods: clock and data. The clock half-period always begins with a transition from low to high or from high to low. The data half-period makes a transition for one value and no transition for the other value. Differential Manchester is self-synchronizing since there is a change in the signal at least every two bits. It is not necessary to know the polarity of the sent signal since the information is not kept in the actual values of the voltage but in their change: In other words, it does not matter whether a logical 1 or 0 is received, but only whether the polarity is the same or is different from the previous value; this makes synchronization even easier. Finally, if the high and low states have the same voltage with opposite polarity, coded signals have zero average DC voltage, thus reducing the necessary transmitting power and minimizing the amount of electromagnetic noise produced by the transmission line. All these positive aspects are achieved at the expense of doubling clock frequency.

`<hdlc_baud>` used to select baud rate

`<hdlc_clk_src>` defines where receiver and transmitter receive clock from.

```
#define SL504_HDLC_FLAG_RXC_RXCPIN  (1L<<0) // RxClk from
                                           //RxC pin (default)
#define SL504_HDLC_FLAG_RXC_DPLL    (2L<<0) // RxClk from DPLL
#define SL504_HDLC_FLAG_RXC_BRG     (3L<<0) // RxClk from BRG0

#define SL504_HDLC_FLAG_TXC_BRG     (1L<<2) // TxClk from
                                            //BRG0 (default)
#define SL504_HDLC_FLAG_TXC_DPLL    (2L<<2) // TxClk from DPLL
#define SL504_HDLC_FLAG_TXC_RXCPIN  (3L<<2) //TxClkfromRxC pin

#define SL504_HDLC_FLAG_DPLL_DIV8   (1L<<4) // DPLL divider 8
#define SL504_HDLC_FLAG_DPLL_DIV16  (2L<<4) // DPLL divider 16
```

There are three main sources of the clock available for both transmitter and receiver. Both of them can be programmed to use any of these clocks.

The first source is the baud rate generator, which is a part of the Zilog Z16C32 chip. It can be programmed to generate a clock of a certain frequency derived from the clock synthesizer chip. There are three main clocks pre-selected for operations: 14.745600MHz, 15.991223MHz and 20.002039MHz. Firmware minimizes error between requested and actual clock by selecting the best combination of clock and divider.

The second source is the RxC pin. RxC pin is a standard clock source for reception when encoding without embedded clock is used (NRZ). The RxC pin can be also a source of Tx clock.

The third source is a DPLL clock extracted from the data line transmission when encoding with embedded clock is used (Biphase encoding).

`SL504_HDLC_FLAG_DPLL_DIV8` and `SL504_HDLC_FLAG_DPLL_DIV16` are used to select a DPLL divider lower than the standard one of 32. DPLL is used in the clocking scheme when RxD provides data without RxC. In this case, BRG1 is programmed as a clock source and RxC is recovered from RxD and used as RxC and (optionally) TxC clock. DPLL mode in HCR is dependent on the encoding used.

`<hdlc_crc_mode>` defines CRC mode used with HDLC protocol:

```
#define SL504_HDLC_CRC_NONE       (0L<<0)   // CRC
// is not checked  for transmit or receive
#define SL504_HDLC_CRC_USER       (1L<<0)   // User
// is responsible for inserting and  checking CRC
#define SL504_HDLC_CRC_16_CCITT   (2L<<0)   // 16-
// bit CCITT CRC is used
#define SL504_HDLC_CRC_16         (3L<<0)   // 16-
// bit polynomial CRC is used
#define SL504_HDLC_CRC_32         (4L<<0)   // 32-
// bit Eth CRC is used
```

16-bit CCITT CRC uses the $(x^{16}+x^{12}+x^5+1)$ polynomial.

16-bit ANSI CRC uses the $(x^{16}+x^{15}+x^2+1)$ polynomial.

32-bit CRC uses $(x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1)$.

HDLC protocol allows filtering of the received packets by its `<hdlc_flt_mode>` which selects the type of filtering to use (selection of the type of filtering is application dependent)

```
#define SL504_HDLC_FLT_NONE       (0L<<0)   // No filtering
#define SL504_HDLC_FLT_A_16       (1L<<0)   // +16
// bits into RxFIFO if Addr matches or B/C as 2 bytes
#define SL504_HDLC_FLT_A_24       (2L<<1)   // +24
// bits into RxFIFO if Addr matches or B/C as 3 bytes
#define SL504_HDLC_FLT_A_32       (6L<<1)   // +32
// bits into RxFIFO if Addr matches or B/C as 4 bytes
#define SL504_HDLC_FLT_EA_LS      (7L<<1)   //
// Places bytes while LS==0, then byte with LS==1 then 16 bits
// as 2 bytes into RxFIFO if EA matches or B/C
#define SL504_HDLC_FLT_EA_24      (3L<<1)   //
// Places 24 bits as 3 bytes into RxFIFO if EA matches or B/C
#define SL504_HDLC_FLT_EA_MS      (11L<<1)  //
// Places bytes while MS==0, then byte with MS==1 then 8 bits
// as 1 byte into RxFIFO if Ext Addr matches or B/C
```

© Copyright 2012
United Electronic Industries, Inc.

```
#define SL504_HDLC_FLT_EA_MS16    (15L<<1)  //
// Places bytes while MS==0, then byte with MS==1 then 16 bits
// as 2 bytes into RxFIFO if Ext Addr matches or B/C
```

The <hdlc_filter> field contains an 8-bit filter address value, if filtering is allowed in <hdlc_flt_mode>.

HDLC protocol can operate with different preamble sizes defined in <hdlc_prmbl_sz> if preamble is selected in <hdlc_preamble> field.

```
#define SL504_HDLC_PRMBSZ_16      (0L<<0)   // 16-
// bit preamble is used
#define SL504_HDLC_PRMBSZ_32      (1L<<0)   // 32-
// bit preamble is used
#define SL504_HDLC_PRMBSZ_64      (2L<<0)   // 64-bit
// preamble is used
```

The HDLC preamble pattern is programmed in the <hdlc_preamble> field and can be one of the following, which is allowed by the protocol. In most cases, no preamble is used.

```
#define SL504_HDLC_PRMB_NONE      (0L<<0)   // no preamble
#define SL504_HDLC_PRMB_ZERO      (1L<<0)   // all zeroes
#define SL504_HDLC_PRMB_ONE       (2L<<0)   // all ones
#define SL504_HDLC_PRMB_FLAG      (3L<<0)   // all flags
#define SL504_HDLC_PRMB_10        (4L<<0)   // alter
//nating 1 and 0
#define SL504_HDLC_PRMB_01        (5L<<0)   //
//alternating 0 and 1
```

HDLC is different from many other protocols in the fact that it continuously transmits idle characters when there is no data to transmit. Idle character can be selected from one of the following:

```
#define SL504_HDLC_IDLE_FLAG      (0L<<0)   //
// continuous flags (0x7E)
#define SL504_HDLC_IDLE_ZERO      (1L<<0)   //
// continuous zeroes
#define SL504_HDLC_IDLE_ONE       (2L<<0)   //
// continuous ones
#define SL504_HDLC_IDLE_MARK      (3L<<0)   // idle
// chars are marks
#define SL504_HDLC_IDLE_SPACE     (4L<<0)   // idle
// chars are spaces
#define SL504_HDLC_IDLE_MS        (5L<<0)   //
// alternating Mark and Space
#define SL504_HDLC_IDLE_01        (6L<<0)   // a
// lternating 0 and 1
```

Most implementations use flag (0x7E) as an idle character.

© Copyright 2012
United Electronic Industries, Inc.

When asynchronous mode is selected, HDLC related fields should be zero and vice versa.

In asynchronous mode, use the following fields to program mode of operation:

<async_baud> - select baud rate (2400 baud to 256kbaud)

<async_char_sz> - select number of bits before parity bit (from 5 to 8), if 9 bit is selected, it is inserted after parity bit. 9-bit asynchronous mode sometimes used in multi-drop protocols to distinguish between device address and device data.

<async_start> - number of start bits (=0 is default means 1 bit)

<async_stop> - number of stop bits (=0 is default means 1 bit)

<async_parity> - selects whether to insert a parity bit after the data bits and

```
#define DQ_SL504_PARITY_NONE    (0L<<0)   // none (parity field
//is not  populated)
#define DQ_SL504_PARITY_EVEN    (1L<<0)   // =1 if sum of data
//bits is even
#define DQ_SL504_PARITY_ODD     (2L<<0)   // =1 if sum of data
//bits is odd
#define DQ_SL504_PARITY_SPACE   (3L<<0)   // always space
//(constant 0)
#define DQ_SL504_PARITY_MARK    (4L<<0)   // always
// mark (constant 1)
```

Fields:

<async_msglen> - watermark level defines message length and

<async_tout> - timeout forces to transmit everything accumulated (in ms) are reserved for messaging and asynchronous event modes

Configuration should be set for each channel regardless of whether it is used for Rx or Tx or both.

3.3.1 Enabling Operations

The function DqAdv504Enable(handle, device, chan_mask) is used to enable and disable operations on the channel. If the bit in the channel mask (1<<channel_number) is “one”, the channel is enabled. If it is zero, the channel is disabled.

© Copyright 2012
United Electronic Industries, Inc.

3.3.2 Sending and Receiving HDLC Frames

`DqAdv504SendFrame()` is used to send a frame and `DqAdv504RecvFrame()` is used to receive a frame.

Both functions have similar parameters:

```
int DqAdv504SendFrame(int hd, int devn, int chnl, int flags, uint8
*data, int rq_size, int *written, int *available)
```

```
int DqAdv504RecvFrame(int hd, int devn, int chnl, int flags, uint8
*data, int rq_size, int *received, int *available)
```

int hd - handle to the IOM

int devn - device number

int chnl - device channel

int flags - behavior modification flags

uint8 *data - data to send

int rq_size - number of bytes to write or size of the receive buffer

int* written (or received) - number of bytes written or received or error code

int* available - number of frame entries left available

Both functions are working with one frame at a time.

Internally, the Z16C32 is programmed to take advantage of DMA operations between its bus and the PSRAM chip on the layer. By default, sixteen 4096 bytes frames are allocated for each channel, for transmit and receive separately. The maximum number of frames is 256.

On transmission, a new frame is added to the list of filled frames and the number of empty frames is returned. The firmware stops accepting new frames when ¾ of the frames are used. At this point DqAdv504SendFrame() returns zero in the <written> field. Please notice when `SL504_PHY_NOECHO` flag is used, the number of Tx frames is limited to one. With this flag, firmware disables receiver from the moment the frame is sent to the moment DMA informs firmware via interrupt that the transmission is completed.

On reception, receiver stops when all frames are used up.

3.3.3 Sending and Receiving Asynchronous Data

`DqAdv504SendMessage()` is used to send a frame and `DqAdv504RecvMessage()` is used to receive a frame.

Both functions have similar parameters:

```
int DqAdv504SendMessage(int hd, int devn, int chnl, int flags,
uint8 *data, int rq_size, int *written, int *available)
```

```
int DqAdv504RecvMessage(int hd, int devn, int chnl, int flags,
uint8 *data, int rq_size, int *received, int *available)
```

int hd - Handle to the IOM

int devn - device number

int chnl - device channel

int flags - behavior modification flags

uint8 *data - data to send

int rq_size - number of bytes to write or size of the receive buffer

int* written (or received) - number of bytes written or received or error code

int* available - number of bytes left available

In the current implementation, the size of transmit and receive FIFOs is 32 characters.

3.3.4 Obtaining Link Status

To obtain the link status, the function
`int DqAdv504GetStatus(handle, device, pSL504_INT_STAT status)`
is used to retrieve accumulated link statistics.

```
// Interface status
typedef struct {
    int cts;        // number of CTS transitions
    int dcd;        // number of DCD transitions
    int tx;         // bytes transmitted
    int ftx;        // frames transmitted
    int rx;         // bytes received
    int frx;        // frames received
    int frm_err;    // frame errors
    int ovr_err;    // overrun errors (Rx)
    int und_err;    // underrun error (Tx)
    int prt_err;    // parity errors
    int tx_abort;   // Tx frame aborts
    int rx_abort;   // Rx frame aborts
    int short_err;  // Rx frames too short to be valid
    int long_err;   // Rx frames too long (>4096) to be valid
    int lines;      // current line state
    int err_stat;   // most recent error status
    int brk;        // number of breaks
    int exithunt;   // ditto exited hunt mode
    int rxidle;     // ditto idle
    int rsv3;
    int rsv4;
    int rsv5;
    int rsv6;
    int rsv7;

    // there are a few important registers to return for debug
purposes
    uint16 tdmr;    // Tranmit DMA status reguster
    uint16 rdmr;    // Receive DMA status reguster
    uint16 ccsr;    // Channel commmand/status register
    uint16 tcsr;    // Transmit commmand/status register
    uint16 rcsr;    // Receive commmand/status register
    uint16 rsv10;
    uint16 rsv11;
    uint16 rsv12;

} SL504_INT_STAT, *pSL504_INT_STAT;
```

© Copyright 2012
United Electronic Industries, Inc.

Most fields are either collected when Z16C32 receives or sends frames or read directly from the chip registers.

The following status bits are useful in the *Transmit Command/Status Register*:

Bit	Name	Description of TCSR bit
7	PreSent	Transmitter has finished sending preamble
6	IdleSent	Transmitter has sent idle condition
5	AbortSent	Transmitter has sent Abort
4	EOF	Transmitter has sent end-of-frame
3	CRCSent	Transmitter has sent CRC code
2	AllSent	Last frame bit has gone out of transmitter
1	TxUnder	Transmitter has encountered underrun condition (starts sending idle character by default)
0	TxEmpty	TxFIFO is empty

The following status bits are useful in the *Transmit DMA Register*:

Bit	Name	Description of TDMR bit
7	Cont	Firmware has issued a Start/Continue command after loading next buffer address
6	GLink	The channel DMA is reading next address in the linked list
5	BUSY	The channel is operating, DMA waits to send/ continues to send data to the transmitter
4	INITG	The channel DMA is fetching information from the linked list or stopped while doing so
3	EOL	The channel DMA has reached the end of the linked list, there is no more data to transfer
2	EOB	The channel DMA has finished sending current frame data
1	HAbort	The channel stopped due to the firmware issued Abort
0	SAbort	The channel stopped due to the Abort command

The following status bits are useful in *Receive DMA Register*:

Bit	Name	Description of RDMR bit
7	Cont	Firmware has issued a Start/Continue command after loading next buffer address
6	GLink	The channel DMA is reading next address in the linked list
5	BUSY	The channel is operating, DMA waits to receive/ continues to receive data from the receiver
4	INITG	The channel DMA is fetching information from the linked list or stopped while doing so

© Copyright 2012
United Electronic Industries, Inc.

3	EOL	The channel DMA has reached the end of the linked list, there are no more buffers to store data
2	EOB	The channel DMA has finished receiving current frame data or frame is too long and the end of buffer is reached
1	HAbort	The channel stopped due to the firmware issued Abort
0	SAbort	The channel stopped due to the Abort command

The following status bits are useful in *Channel Command/Status Register*:

Bit	Name	Description of CCSR bit
15	RCCF Overflow	RCC FIFO Overflow (should not occur if DMA is working properly and frames are read on time)
14	RCCF Avail	RCC FIFO has data (DMA takes care of emptying the FIFO)
12	DPLL Sync	DPLL is in sync with the input clock embedded in RxD
11	DPLL 2Miss	DPLL has seen two consecutive missing clocks
10	DPLL 1Miss	DPLL has seen a missing clock

3.3.5 Aborting Transmission

Finally, to abort a HDLC frame transmission, you can call:

int DqAdv504AbortTx(int hd, int devn, int channel, uint32* status)

This function issues abort commands to the transmitter and clears all frames of the existing data. The status returned is the content of Z16C32 TCSR register.

Appendix A

A. Accessories

The following cables and STP boards are available for the SL-504 layer.

DNA-CBL-62

This is a 62-conductor round shielded cable with 62-pin male D-sub connectors on both ends. It is made with round, heavy-shielded cable; 2.5 ft (75 cm) long, weight of 9.49 ounces or 269 grams; up to 10ft (305cm) and 20ft (610cm).

DNA-STP-62

The STP-62 is a Screw Terminal Panel with three 20-position terminal blocks (JT1, JT2, and JT3) plus one 3-position terminal block (J2). The dimensions of the STP-62 board are 4w x 3.8d x1.2h inch or 10.2 x 9.7 x 3 cm (with standoffs). The weight of the STP-62 board is 3.89 ounces or 110 grams.

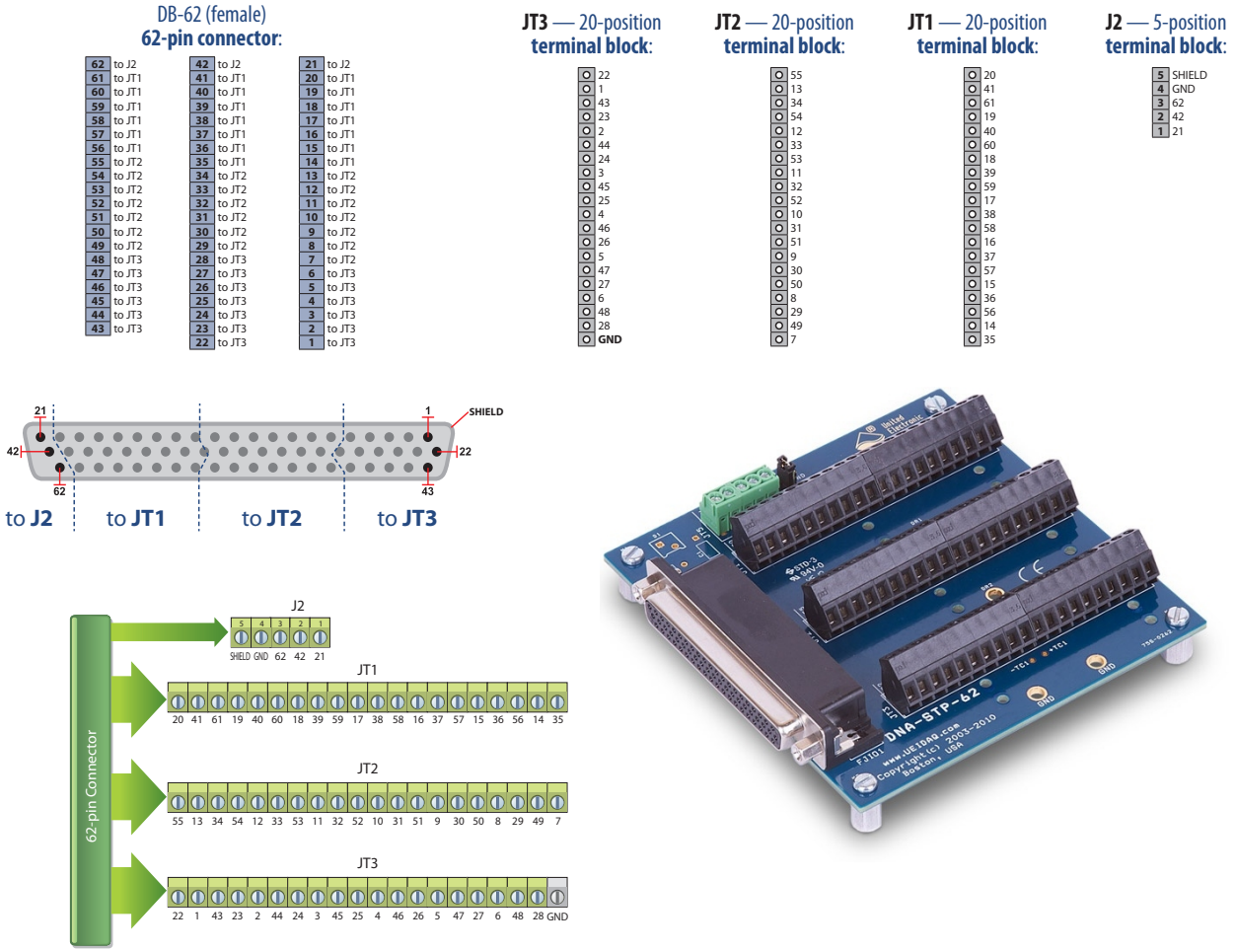


Figure A-1. Pinout and photo of DNA-STP-62 screw terminal panel

Index

B
Block diagram 12

C
Cable(s) 35
Capabilities 13
Cleaning-up the Session 21
Cleaning-up the session 21
Configuring the Resource String 16
Conventions 2
Creating a Session 16

H
High Level API 16

I
Isolation 3

J
Jumper Settings 15

L
Layer Position Jumper Settings 15
Low-level API 22

O
Organization 1

S
Screw Terminal Panels 35
Setting Operating Parameters 4
Support ii

W
Wiring 14

© Copyright 2012
United Electronic Industries, Inc.